Journal of Theoretical and Applied Information Technology
15th December 2020. Vol.98. No 23

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

NEW EFFICIENT AND ROBUST NTRU POST-QUANTUM KEY EXCHANGE RELEASE "NTRU-ROBUST"

[1] El HASSANE LAAJI, [1]ABDELMALEK AZIZI
[1]Mohammed First University Oujda Morocco
E-mail: e.laaji@ump.ac.ma, abdelmalekazizi@yahoo.fr

ABSTRACT

Since 2016, the National Institute for Standards and Technology (NIST) launched a post-quantum standardization project. Now the competition reaches its third round, and NIST inviting the cryptographic community to participate by improving and analyzing the finalist candidates. On our side, we contribute by creating an improved release of NTRU Lattice-Based post-quantum key exchange (KEM) scheme, called "***NTRUrobust"***; with parameters set that meet the category 5 security level defined by the NIST. ***NTRUrobust*** used our own Fast Modular Multiplication algorithm (FMMA) and the Number Theoretic Transform algorithm (NTT) together for speeding-up the polynomials multiplication in the cryptographic process. We obtain drastic results; our release is faster by factor up to 93 times than ***NTRUphs821*** release proposed by the NTRU team. ***NTRUrobust*** warrants strong security level, perfect correctness, and great flexibility for eventual future extensions.

Keywords: *Post Quantum cryptography, Modular Multiplication, NTRU, NTT, KEM.*

1. INTRODUCTION

The cryptography is omnipresent in our everyday life though many applications in order to secure sensitive data exchanged or stored in electronic devices. The most current cryptographic protocols are based on RSA, Diffie&Hellman (DH), or elliptic curve cryptography (ECC). Unfortunately, all these cryptosystems will be vulnerable to quantum computer attacks.

It is why, in 2016 the National Institute of Standard and Technology (NIST) lunch a post-quantum cryptosystem standardization project [1] for choosing one or more post-quantum cryptosystem able to resist quantum computer attacks.

Now, the NIST competition reaches its third round by selecting seven finalist candidates: "The third-round finalist public-key encryption and key-establishment algorithms are Classic McEliece, CRYSTALS-KYBER, NTRU, and SABER. The third-round finalists for digital signatures are CRYSTALS-DILITHIUM, FALCON, and Rainbow" as announced in [2].

Except the Classic McEliece and Rainbow, all algorithms are lattice-based schemes and NIST states in its latest report, that "The structured lattice schemes appear to be the most promising general-purpose algorithms for public-key encryption/KEM and digital signature schemes "[2].

1.1 Related works

NTRU team presented four KEM (Key Exchange Mechanism) releases that their parameters sets meet the categorization based on security levels 1, 3, and 5 as defined by NIST. All those releases operate in the polynomials ring of the form $R_q = Z_q[X]/(X^n - 1)$, with modulus $\boldsymbol{q}$ is power of two and $\boldsymbol{n}$ prime number. The parameters sets of NTRU releases are:

- ✓ NTRUhps2048821 with parameters set that meets category 1; it is equivalent to an algorithm that is at least as hard to break AES128;
- ✓ NTRUhps2048677 with parameters set that meets category 3; it is equivalent to an algorithm that is at least as hard to break AES192.
- ✓ NTRUhps4096821 with parameter set that meets category 5; it is equivalent to an algorithm that is at least as hard to break AES256.
- ✓ And the NTRUhrss701 release which is presented as variant of NTRUhps2048677.

In term of the security, NTRU is based on stronger lattices assumption, it has resisted for 20 years of cryptanalysis. The NIST shows that NTRU assumes an interesting security levels, and states that NTRU "lacks a formal worst-case-to-average-

case reduction", and it is more efficient than others Lattice-based cryptosystems [2, 3].

But in term of the performance, NIST states that NTRU submission is "not quite at the level of the highest-performing lattice schemes"[2].

Therefore the goal of our contribution is to dress this weakness of NTRU performance, by developing new efficient and robust a NTRU version, with high confidence in its security.

1.2 Contributions

We note that, this work is an extended work of our latest paper titled "Fast Modular Multiplication algorithm applied to a Ring-LWE scheme"[4], which describes the creation of an improved release of post-quantum key exchange "NewHope" based on Ring-LWE problem [5].

Along with the cryptographic community, we contribute by creating a new efficient and robust NTRU post-quantum key exchange (KEM) release called "NTRUrobust", with parameters set {n=1024,q=65537,p=2} that achieving the category 5 security level , and it can be a variant of NTRUhps4096821 with parameters set {q=4096, n=821, p=3} cited above.

Generally, the execution time of the NTRU based protocols is dominated by the multiplications in polynomials ring.

Therefore, for speeding-up the polynomials multiplication in the cryptographic process (Keys Generation + Encapsulation + Decapsulation), our NTRUrobust implementation uses NTT algorithm [6,7], combined with our Fast modular multiplication algorithm (FMMA), which represent a variant of Montgomery algorithm[8] and it is two times faster [4].

The benchmarking result of our release compared to NTRUhps4096821 release shown that the performance of our release is about 93 times faster, and warrants a stronger security level, and perfect correctness.

We also obtained a good result by comparing NTRUrobust to CRYSTAL-KYBER and SABER, which their parameters meet category 5 security levels defined by NIST, as we will see in section 6.

We note, also that our release used the latest KECCAK hash function which has recently been standardized as SHA-3 in FIPS202 [9],

1.3 Outline

The rest of the paper is organized as follows: the section.1 contains this introduction; Section 2, recalls the background of the Lattices-Based-Cryptography, and briefly describes the FMMA and NTT algorithms; the Section 3, presents a description of some related works based on structured Lattice scheme; in Section 4, we present NTRU historic and a description of NTRUhps post-quantum cryptosystem. The proposed scheme NTRUrobust and the corresponding efficiency and security analysis are presented in Sections 5, 6 and 7. Finally a conclusion is provided.

2. BACKGROUND

In this section, we will focus to provide only descriptions of the principal subjects that are evoked in this work. So, we give a brief definition of the lattice-based cryptography, and we describe our Fast Modular Multiplication and the improved NTT algorithms.

2.1 Lattice-Based cryptography

The Lattice-Based-Cryptography is defined by Hoffstein et.al in [10] as follow:

The Lattice $L_{(B)}$ *of* $\mathbb{R}^n$ is the set of vectors $\vec{v}$ generated by the basis $B=(\vec{e_1},...,\vec{e_n})$ with all vector coefficients are integer numbers in $\mathbb{Z}$, formally:

$$L_{(B)}=\{\vec{v}\in\mathbb{R}^n,(a_1,...,a_n)\in\mathbb{Z}^n \text{ and } \vec{v}=a_1\vec{e_1}+...+a_n\vec{e_n}\}.$$

The Lattice Cryptography is based on the complexity to break Lattices Cryptosystems by posing problems that are hard to solve. The principal Lattice problems are SVP and CVP and their definitions are as follow:

2.1.1 The Shortest Vector Problem (SVP)

Finding (SVP) in Lattice $L_{(B)}$ is finding a non-zero vector that minimizes the Euclidean norm. Formally the problem SVP is to find a non-zero vector:

$$\vec{v}\in L_{(B)} \quad \forall\vec{x}\in L_{(B)} \text{ we have } \|\vec{v}\| \leq \|\vec{x}\| \qquad (1)$$

2.1.2 The Closest Vector Problem (CVP)

Given the Lattice $L_{(B)}$, and a vector $\vec{w}\in\mathbb{R}^m$ to find a vector $\vec{v}\in L_{(B)}$ "Closest" to $\vec{w}$, is to find a vector $\vec{v}\in L_{(B)}$ that minimizes the Euclidean norm $\|\vec{w}-\vec{v}\|$ where:

$$\min\{\|\vec{w}-\vec{v}\|\}\rightarrow \text{CVP}=\vec{v}. \qquad (2)$$

Many others approximate problems of SVP and CVP exist, like uSVP, CoreSVP[5], etc. The CoreSVP is the cost of lattice attacks of one call to solve a SVP with block size ***b,*** by using BKZ Lattice reduction algorithm, as we will describe in section 6 [11].

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2.2 Fast Modular Multiplication Algorithm

The motivation for studying high-speed algorithms for modular multiplication comes from their applications in some cryptosystems. FMMA is a method to carry out a fast modular multiplication. It is two about times faster than the Montgomery algorithm, and it can be its variant [4].

FMMA is constructed specially for Fermat prime numbers and all numbers of the form $q = 2^k + 1$. In our case study, we use the fourth Fermat prime number as the modulus for our implementation, as we will describe below [12].

The principle of our algorithm is to transform the modulation of a prime number to the number of power of two. It allows the reduction and the modular multiplication to be fast and more efficient on general-purpose computers, signal processors, and microprocessors. The reduction (%), the multiplication (*), and the division (/) will be respectively replaced by logic operators AND (&), Shift (<<), and Shift (>>).

2.2.1 Algorithm 1: FMMA

Input: *Integers* x, y, *modulus* $q = 2^k + 1$, *and* $\phi = (q-1)$.

FMMA():

1. $p \leftarrow x * y;$
2. $z \leftarrow p \,.\&\, (\phi - 1);$
3. $d \leftarrow (p - z) >> log_2(phi);$
4. $result \leftarrow (z - d);$
5. *if* $(result < 0)$ *then return* $result + q.$
6. *else return result.*

Output: Reduced number: $result = x * y \pmod q$.

2.3 Number Theoretic Transform (NTT)

The number-theoretic transform (NTT) is a generalization of the Discrete Fourier Transform (DFT), see [6,7], which is carried out in positive Integer group and finite fields whereas the DFT is defined in complex numbers group.

The Number Theoretic Transform (NTT) provides efficient polynomials multiplication in the ring of the form $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ (with n power of two and q prime number). NTT has many applications in computer arithmetic and cryptographic domain, because it reduces the time complexity from $O(n^2)$ *to* $O(n * log(n))$.

To use NTT algorithm we must choosing the modulus that satisfying, $q = kn + 1$ then the multiplicative group $\mathbb{Z}_q^n$ has size $\varphi(q) = q - 1 = k.n$ and a generator g, and computing the primitive nth root of unity Omega ω:

$$\omega = g^k \pmod q \text{ and } \omega^n = g^{kn} = g^{\varphi(q)} = 1 \pmod q. \quad (3)$$

2.3.1 Transforming a polynomial from Normal form to NTT form

For a polynomial $f = \sum_{i=0}^{n-1} f_i X^i \in R_q$ the NTT function is defined by:

$$NTT(f) = fNtt = \sum_{i=0}^{n-1} f Ntt_i X^i. \quad (4)$$

$$with \quad fNtt_i = \sum_{j=0}^{n-1} \gamma^j f_j \omega^{ij} \pmod q. \quad (5)$$

Where Gamma $\gamma = \sqrt{\omega}$ is the 2nd root of unity.

2.3.2 Transforming a polynomial from NTT form to Normal form

The inverse of NTT function to return to normal form is computed by the below formula:

$$invNTT(fNtt) = f = \sum_{i=0}^{n-1} f_i X^i. \quad (6)$$

$$with \quad f_i = n^{-1}\gamma^{-i} \sum_{j=0}^{n-1} f Ntt_j \omega^{-ij} \pmod q. \quad (7)$$

So the NTT algorithm can perform the multiplication of two polynomials $h = f * g \in R_q$; by transforming them to NTT form (fNTT and gNTT); computing the product in NTT form by the point-wise multiplication noted by $(\otimes)$ $hNTT = fNTT \otimes gNTT \bmod q$ (that means we obtain $hNtt_i = fNtt_i * gNtt_i \bmod q$); and finally transforming the $hNTT$ polynomial from NTT form to normal form by the inverse of NTT function: $h = invNTT(hNTT)$.

Consequently, an important reduction cost of multiplication can be achieved by pre-computing and storing the powers values related to the parameters: ω *and* $\gamma = \sqrt{\omega}$ [6].

2.3.3 Pre-computed arrays.

For using the improved NTT algorithm, based on the Cooley-Tukey (CT) buttery efficiency, we should computing and storing five arrays: **Bitrev**, **Omega**, **InvOmega**, and **Gama** in bit-reversible order, and **invGama** in normal order as follow :

- ✓ **Bitrev** : Stores the value from 1 to (n-1) in bit-reversible;
- ✓ **Omega**: Stores the powers of ω in bit-reversible: $Omega[i] = \omega^{Bitrev[i]} \pmod q$.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

- ✓ **InvOmega**: Stores the powers of ω^{-1} in bit-reversible: $InvOmega[i] = \omega^{-Bitrev[i]} \pmod q$.
- ✓ **Gama**: Stores the powers of γ in bit-reversible: $Gama[i] = \gamma^{Bitrev[i]} \pmod q$.
- ✓ **InvGama**: Stores the powers of γ^{-1} in normal order and multiplying each coefficient by the inverse of $n \pmod q$:

$$invGama[i] = \frac{1}{n} * \gamma^{-[i]} \pmod q.$$

2.3.4 Algorithm 2: improved NTT algorithm

Input : a polynomial ***f***, a pre-computed array ***precomp***, dimension ***n***, and modulus $q = k * n + 1$.

Function : CoolyNTT()

1. $for(i = 0; i < 10; i += 2)\ do:$
// *Even Level*
2. $d = (1 << i);$
3. $for(st = 0;\ start < d; st++)\ do:$
4. $move = 0;$
5. $for(j = st;\ j < n-1;\ j += 2 * d)\ do:$
6. $A[j] = (A[j] + A[j + d]);$
7. $A[j+d] = \text{FMMA}(Precomp[move++], (A[j] - A[j+d]), q);$
8. *end for (line 5);*
9. *end for (line 3);*

// Odd Level

10. d <<=1;
11. $for(st = 0;\ st < d; st++)\ do:$
12. $move = 0;$
13. $for(j = st;\ j < n-1;\ j += 2 * d)\ do:$
14. $A[j] = (A[j] + A[j + d]);$
15. $A[j+d] = \text{FMMA}(Precomp[move++], (A[j] - A[j+d]), q);$
16. *end for (line 11);*
17. *end for (line 13);*
18. *end for (line 1);*

End algorithm.

Output: a *fNtt=CoolyNTT(f)* in bit-reversed ordering.

Comment: This principal *CoolyNTT(f)* will be called by the algorithm 3 and the algorithm 4. The ***Precom*** array in line 7 and 15, takes the value of ***Gama*** or ***invGama*** **(**as described above) according to our need to compute the forward NTT of the polynomial ***f*** or its NTT inverse, and this principal algorithm calls the FMMA algorithm in line 7, and line 15 for reducing each multiplication *modulo q*.

We note this version of NTT is inspired from [5, 7].

2.3.5 Algorithm 3: Transforming a polynomial from normal form to NTT form

Input: a polynomial ***f, n,*** pre-computed array ***Omega*** and ***Gama*** as are described above.

Function : NTTfunc(f)

1. $bitrev\_vector(f);$
2. $for\ (int\ i = 0;\ i < n;\ i++)\ do:$
3. $f[i] \leftarrow \text{FMMA}(f[i],\ Omega\ [i], q);$
4. *end for (line 2);*
5. $fNtt \leftarrow CoolyNTT(f,\ Gama);$
6. Return *fNtt*;

Output: a polynomial ***fNtt*** in NTT form and in normal order.

Comment: This function calls the FMMA algorithm with pre-computed array ***Omega*** as argument (in line 3), and calls ***CoolyNTT*** algorithm with pre-computed array ***Gama*** as argument (line 5), and itself is called for each polynomials multiplication by the Keys generation, Encapsulation, and Decapsulation algorithms.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

2.3.6 Algorithm 4: Transforming a polynomial from NTT form to normal form

Input: polynomial ***r, n,*** pre-computed arrays ***invOmega*** and ***invGama*** containing respectively the power of $\omega^{-i} (\text{mod}\, q)$ in bit-reversible, and $\gamma^{-i} (\text{mod}\ q)$ in normal order as are described above.

Function : invNTTfunc(fNtt)

1. $bitrev\_vector(fNtt)$;
2. $f \leftarrow CoolyNTT(fNtt, \text{invOmega})$;
3. $for\ (int\ i = 0\ ;\ i < n\ ;\ i++)\ do$:
4. $f[i] \leftarrow \text{FMMA}(f[i]\ ,\ \text{invGama}\ [i], q)$;
5. $end\ for\ (line\ 2)$;
6. Return r;

Output: A polynomial f in normal form and in normal order.

Comment: The same, this function calls *CoolyNTT* the algorithm with pre-computed array ***invOmega*** as argument (in line 2) and calls the FMMA algorithm with pre-computed array ***invGama*** as argument (in line4), and itself is called for each polynomials multiplication by the Keys generation, Encapsulation, and Decapsulation algorithms.

3. RELATED WORKS

Many others works used one or more algorithms for increasing the performance of the multiplication and modular multiplication. In this section we give small description of some lattice-based schemes that use one or more algorithms commonly in their cryptographic process to increase the performance of the multiplication and modular multiplication like NTT algorithm, Montgomery algorithm, Karatsuba algorithm etc.

Specially, we describe briefly the two finalist candidates CRYSTAL-KYBER and SABER, which are competitors of NTRU. Also, we give a small description of FALCON which is a lattice-based signature scheme, and it is based on the NTRU assumption.

3.1 CRYSTALS-KYBER

The CRYSTALS-KYBER is post-quantum key encapsulation mechanism (KEM) constructed and based on the hardness of the Module Learning With Errors (MLWE) problem, and inspired from Learning With Error (LWE) created by Regev since 2008[13][14]. It is an IND-CCA2-secure KEM constructed from IND-CPA security by using Fujisaki–Okamoto transformation [15].

The KYBER scheme operates in the ring of the form $R_q = Z_q[X]/(X^n + 1)$ with the dimension ***n*** is power-of-two and the modulus ***q*** is prime number. For speeding-up the polynomials multiplication and increasing its performance, KYBER uses the number theoretic transform (NTT). NIST states that "The scheme has excellent all-around performance for most applications"[2].

We note that KYBER team defined a release ***Kyber1024*** with the parameters set meets the category 5 security level.

For more details about KYBER the reader can see [16] in NIST website.

3.2 SABER

SABER is lattice-based (KEM) scheme constructed and based on the hardness of the Module Learning With Rounding (MLWR) problem [17]. Like KYBER, it is an IND-CCA2-secure KEM constructed from IND-CPA security by using Fujisaki–Okamoto transformation [15].

The SABER cryptosystem operates in the polynomials ring of the form $R_q = Z_q[X]/(X^n - 1)$, with ***n*** power of two and modulus ***q*** power of two. This domain not allows us to use efficiency the NTT algorithm.

The SABER team states that “the rounding operation and power-of-2 modulo in SABER allows for the efficient optimization of the modular reduction and polynomial multiplication steps”.

We note that SABER team defined a release ***FireSaber*** with the parameters set meets the category 5 security level [18].

NIST experts judge that "SABER has excellent performance and would be immediately suitable for general-purpose applications, and it is one of the most promising KEM schemes to be considered for standardization at the end of the third round"[2].

3.4 FALCON

FALCON [19] is a lattice-based signature scheme, its security is based on NTRU assumption, and it operates in the polynomials ring of the form $R_q = \square_q[X]/(X^n + 1)$ with ***q*** prime number and ***n*** power of two.

The same, FALCON used the NTT algorithm with Montgomery algorithm for speeding up the polynomials multiplication and modular multiplication.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

NIST expert state that "It is efficient, and offers good performance. So FALCON was selected by NIST to continue in third-round [2].

4. NTRU

In 1996, NTRU was introduced by the three mathematicians J. Hofstein, J. Pipher, and J. H. Silverman and then published in 1998 [10]. It was presented as an alternative to RSA, ECDH and ECC. NTRU releases have also been standardized by the IEEE P1363.1 standard in April 2011, and by the X9.98 standard [3][11].

NTRU is completely constructed from Lattice-Based-Cryptography. Besides the conjectured security against quantum attacks, lattice-based schemes tend to be algorithmically simple and highly parallelizable [2].

Its domain of computation is the polynomials ring of the form $R_q = \mathbb{Z}_q[X]/(X^n - 1)$ with ***n*** prime number and the modulus ***q*** is power of two, or the polynomials ring of the form $R_q = \mathbb{Z}_q[X]/(X^n + 1)$ with ***n*** is power of two and the modulo ***q*** is prime number.

Since its first creation there are several versions, the latest NTRU is now candidate to NIST's post-quantum standardization project, and it is selected among the four finalist public key encryption/KEM schemes.

4.1 NTRU-HPS KEM scheme

The NTRU candidate is a structured lattice-based KEM obtained from by using generic transformation of PKE and it is IND-CCA2 secure. "The NTRU-HPS follows **H**offsein, **P**ipher, and **S**ilverman's use of fixed weight sample spaces" [3]. is It defined in the polynomials ring of the form $R_q = Z_q[X]/(X^n - 1)$ and the sub-ring $S_q = \mathbb{Z}_q[X]/(\Phi_n)$ with $\Phi_n = (x^n - 1)/(x - 1)$ and $\Phi_n = x^{n-1} + x^{n-2} + \ldots + 1$.

The authors presented four versions with different parameters set that warrant the security levels defined and classified by NIST in categories 1,3, and 5 [3].

The releases and their parameter set are: NTRUhps2048509 $\{n = 509, q = 2048, p = 3\}$ for achieving the security level category 1; NTRUhps2048677 $\{n = 677, q = 2048, p = 3\}$ for achieving the security level category 3; NTRUhps4096821 $\{n = 821, q = 4096, p = 3\}$ for achieving the category 5, (and NTRUhrss701 version which is a variant of NTRUhps2048677).

The polynomials are sampled according to uniform distribution with a ***seed*** and used KECCAK[9] ***SHA-3*** hash function implementation, which is the latest standard hash function. They are chosen in the set {-1, 0, 1} with some criteria of number of coefficients equal to (1) and number of the coefficients equal to (-1). The authors designed the: L_r , L_m, L_g, L_f, respectively the sets of the polynomials {*r, m, g, f*} the reader can see for more details [3].

We remark that the use of the ring R_q and sub-ring S_q has no impact on the NTRU security, it is used just for speeding-up the convolution multiplication of in this polynomials sub-ring, and the inverse of the polynomials by using the extended Euclidean algorithm. It is so easy to switch from R_q *to* S_q or the inverse, just by multiplying by $(x-1)$ or dividing by $(x-1)$.

For more information about NTRU description and implementations the reader can find them [3] in NIST website.

4.2 NTRUhps KEM algorithms description

In this subsection we describe briefly the cryptographic algorithms, for more details the reader can see the document in [3].

4.2.1 Keys Generation

The keys generation begin by: (1) generating uniformly two polynomials (f, g) in the sets (L_f, L_g); (2) computing a polynomial F_q, F_3 as the inverses of f respectively in S_q, S_3 ; (3) computing the public key $h = p(g * F_q)$ in R_q; (4) and finally computing the inverse of h in S_q noted H_q.

4.2.2 Encapsulation

The encapsulation begin by: (1) generating randomly a message m (shared key) in L_m set, and hashing it in a string Km ; (2) Generating uniformly a noise r as polynomial in L_r set; (3) computing a cipher-text $c = h * r + m$ in R_q .

4.2.3 Decapsulation

The decapsulation begin by: (1) computing a polynomial $a = f * c$ in R_q ; (2) computing a decrypted message $m = a * F_3$ in R_3 ; (3) computing a polynomial $r = c - m \bmod q$; (4) checking $if\,(r, m) \in L_r \times L_m$ which means that the reconciliation of the shared key is good and returns

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

$(r,m,0)$, the shared key is the hashed string of ***m*** ; else it returns $(0,0,1)$.

According to the NIST experts' analysis NTRU is an exciting field of research, and it is very efficient but it has a small performance gap in comparison to KYBER and SABER. In particular, NTRU has a slower key generation than KYBER and SABER [2].

5. OUR NTRU_ROBUST RELEASE

To correct the performance weakness of NTRU candidate, we contribute by creating an improved release of NTRU, which could be a variant of NTRUhps4096821 release. We obtained drastic result in the term of performance by combining our FMMA algorithm with NTT algorithm as we described above in section 2.

Our release, called "NTRUrobust", is based and inspired from the NTRUhps cryptographic algorithms described in section 4. It is defined in the polynomials ring of the form $R_q = \mathbb{Z}_q[X]/(X^n+1)$ with the dimension $n=1024$ and the modulus $q=65537$ which is the fourth Fermat prime number that allows us to use efficiency the FMMA and NTT algorithms.

In this section, we describe our *NTRUrobust* Lattice-Based (KEM) scheme. And for good illustration of our contribution, we introduce the NTT functions in the keys generation, encapsulation, and decapsulation algorithms.

5.1 Parameters definition method

To use the NTT algorithm combined with *FMMA* algorithm, we define herein the necessary parameters as follow:

- ✓ The multiplicative group $\mathbb{Z}_q$ has a generator g and a size $\varphi(q)=q-1=k.n$, which represents the Euler indicator.
- ✓ And defining the nth primitive root of unity ω and its primitive 2n-th root of unity such that $\gamma=\sqrt{\omega}$, computed by the below formula.

$$\omega = g^k \pmod q \text{ and } \omega^n = g^{kn} = g^{\varphi(q)} = 1 \pmod q.$$

5.2 NTRUrobust parameters set

Our *NTRUrobust* release is defined by the parameters set that achieves the category 5 security level $\{n=1024, q=65537, p=2, \sigma=\sqrt{16/2}\}$, with σ is the standard deviation used for choosing the polynomials according to Centered Binomial Distribution. The modulus used is the fourth Fermat prime number $q=2^{2^4}+1=2^6*1024+1=65537$.

Fermat prime numbers were first studied by Pierre Fermat, for more details the reader can see [12].

For computing the needed parameters values of NTT, we should follow the steps below:

- ✓ The parameters *n=1024, k=2^6=64, and q=64*1024+1=65537* which satisfying the condition: q=kn+1;
- ✓ Finding a generator *g=4441* that satisfying the conditions: g^k *!*=1 and $g^{kn}=1$;
- ✓ Computing n-th primitive root of unity omega:
- ✓ $\omega = g^k \pmod q = 1089$, its square root gamma: $\gamma=\sqrt{\omega}=33$, the inverse of gamma: $\gamma^{-1} \pmod q = 1986$, and the inverse of omega: $\omega^{-1} \pmod q = 11976$
- ✓ The inverse of ***n*** modulo ***q***, $n^{-1} \pmod q = 65473$ which will be used by the inverse of NTT function to transform the polynomials from NTT form to normal form for computing the ***InvGama*** array as defined in subsection 2.3.3.

The particularity of the fourth Fermat prime number is that it has only one root of unit $\omega=1089$.

5.3 Description of NTRUrobust protocol

In this part, we describe the proposed *NTRUrobust* post-quantum key exchange(KEM) inspired by the NTRUhps KEM scheme. And for illustrating our contribution and our improvement, we modified the original algorithms of NTRUhps (keys generation, encapsulation, and decapsulation) by using the NTT functions which includes the *FMMA* function for each multiplication of two coefficients (as described in section 2).

The polynomials are sampled according to Centered Binomial Distribution (sampCBD() function) rather than Uniform Distribution (sampUD() function), Alkim.et.al[5] state that "Sampling from the Centered Binomial Distribution is rather trivial in hardware and software. Additionally, the implementation of this sampling algorithm is much easier to protect against timing attack and does not decrease the security".

The sets of the polynomials {*f, g, r, m*} used by the NTRUrobust protocol are defined respectively as below:

$$L_f = \{f \in R \text{ with } f_i \in [-3,..,3]\} ;$$

$$L_g = \{g \in R \text{ with } g_i \in [-3,..,3]\} ;$$

$$L_r = \{r \in R \text{ with } r_i \in \{0,1\}\} ;$$

$$L_m = \{m \in R \text{ with } m_i \in \{0,1\}\} .$$

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Also, we use SHA3-256 hash function rather than using the SHAKE-256 hash function that allows us to hash a binary polynomial (***m***) directly into string format without coding it into string format before hashing it. For more details, the reader can see [3, 9].

We also note that our version takes the private key F in the form $F = p * f + 1$. This form allows us to avoid the computation of the inverse of $f \pmod p$ because $F = p * f + 1 \pmod p = 1$ [20].

The NTRU authors' shown that the use of private key in this form decrease the rate of the decryption failure, but by the parameters set of our release implementation we obtained a perfect correctness of the decryption by executing the script developed by Hoffstein et.al [21, 22].

The Keys Generation, Encapsulation, Decapsulation algorithms below, describe the protocol operations and illustrate the use of the *NTT* algorithm and the *SHA3-256* hash function. The *FMMA* modular multiplication function is integrated in NTT functions for each coefficients multiplication in all the cryptographic process (as described in the algorithm 2,3, and 4).

5.3.1 Algorithm 5 : Keys Generation

Input : the sequence parameters $\{n, q, p\}$ and a ***seed***.

1. $f, g \leftarrow sampCBD(seed)$;
2. $F \leftarrow F = pf + 1$;
3. $FNtt, gNtt \leftarrow NTTfunc(F, g)$;
4. $invFNtt \leftarrow \frac{1}{FNtt} \pmod q$;
5. $hNtt \leftarrow (gNtt \otimes invFNtt \pmod q)$;
6. $invHNtt \leftarrow \frac{1}{hNtt} \pmod q$;
7. $pk \leftarrow encode(hNtt)$;
8. $sk \leftarrow encode(FNtt, invHNtt)$;

Output : private key ***FNtt*** and ***invHNtt*** encoded in the string ***sk*** and the public key ***hNtt*** encoded in ***pk.***

Comment : In the key generation our implementation generate both private keys ***(f, g)*** in the same time by the sampCBD(seed) function(line 1) that allows us to increase the key generation process, this function uses the SHAKE-256 Keccak hash function. The private keys and the public keys are kept in NTT form and encoded into string ***sk, pk.*** In (line 4) and (line 6), the inverse of polynomials (hNTT, FNtt) are found by computing the inverse of each coefficient modulo ***q***, $invHNtt_i = \frac{1}{hNtt_i} \pmod q$ and $invFNtt_i = \frac{1}{FNtt_i} \pmod q$ by extended Euclidean algorithm.

5.3.2 Algorithm 6 : Encapsulation

Input : The public key ***pk*** and a ***seed***

1. $msg \leftarrow randomly(seed)$;
2. $m \leftarrow encode(msg)$;
3. $(hNtt) \leftarrow decode(pk)$;
4. $mNtt \leftarrow NTTfunc(m)$;
5. $r \leftarrow sampCBD(seed)$;
6. $rNtt \leftarrow NTTfunc(r)$;
7. $cNtt \leftarrow (rNtt \otimes hNtt) + mNtt$;
8. $SSc \leftarrow SHA3-256(m)$; (***SSc*** is the shared key)
9. $C \leftarrow encode(cNtt)$. (Codified in string format)

Output : The ciphertext ***C***.

Comment: in (line 1) the encapsulation function chooses randomly a message and encodes it into binary polynomial, and then in (line 8) hash it by Keccak hash function SHA3-256 to produce the shared key ***SSc***. In (line 9) the cipher-text is transformed from polynomial form to string format.

5.3.3 Algorithm 7: Decapsulation

Intput : The cipher-text ***C*** and the private key ***sk***

1. $(FNtt, invHNtt) \leftarrow decode(sk)$;
2. $(cNtt) \leftarrow decode(C)$;
3. $bNtt \leftarrow cNtt \otimes FNtt \pmod q$;
4. $b \leftarrow InvNTT(bNtt) \pmod q$;
5. $b \leftarrow lefting \quad b_i \in \{\frac{-q-1}{2}, \frac{q-1}{2}\}$;
6. $m \leftarrow b \pmod p$;
7. $mNtt \leftarrow NTTfunc(m) \pmod q$;
8. $rNtt \leftarrow (cNtt - mNtt) \otimes invHNtt \pmod q$;
9. $r \leftarrow InvNTT(rNtt) \pmod q$;
10. $If\ (r, m)\ not \in \{L_r \times L_m\}$ *return False* ;
11. $SSd \leftarrow SHA3-256(m)$;

Output : The shared key ***SSd.***

Comment: In (lin 6) the first step is achieved by computing the decryption message ***m*** and the decapsulation function produce a polynomial ***r*** in (line 9) and then in (line 10), the reconciliation is good, if only if the coefficients of the polynomials ***(m, r)*** are in $\{0,1\}$. That means the shared key ***SSc*** (Hashed string) in the encapsulation algorithm is equal to shared key ***SSd*** (hashed string) in the decapsulation algorithm; else the complete process must be repeated until the reconciliation is exact.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

The decryption failure rate of our *NTRUrobust* release implementation is ZERO that warrants perfect correctness. The result is obtained by using the python script developed by Hoffstein et.al, and executing it in Sage software [21, 22].

6. BENCHMARKING RESULT

In this section we will present the performance results of our *NTRUrobust* compared especially to the performances of *NTRUhps4096821*. And also we present the performance results of our release compared to the performances of *Kyber1024* and *FireSaber* releases respectively of KYBER and SABER and that their parameters sets meets the category 5, for having the coherence comparison.

We note that all implementations are performed on a PC-TOSHIBA with an Intel(R) Core(TM) i7-2630QM CPU, 2 GHz processor, RAM 8 GO, under environment Windows 7-32 bits and Dev-C++ 4.9.9.2.

The implementation of *NTRUrobust* release described in this paper is available on the Google drive website at [23]; and the NTRU, KEYBER, and SABER implementations are available in the NIST website at [24].

6.1 Benchmarking between NTRUrobust & NTRU4096821

In this subsection, we present the performance results of the benchmarking of our NTRUrobust implementation with parameters *{n=1024, q=65537, p=2}* compared to the NTRU4096821 with parameters *{n=821, q=4096, p=3}*. We built the software of both implementations on the same machine cited above, and we report the median result of 100 runs.

The performance results can be found in Table 1, and the result values are given in milliseconds (ms).

Table 1: Speed performance benchmarking (ms)

Schemes	Keys Gen (ms)	Encap (ms)	Decap (ms)
NTRUhps 4096821	206	3.60	8.43 ms
NTRU robust	1.25	0.47	0.62
Speed-up Factor (X)	165 times	8 times	11 times

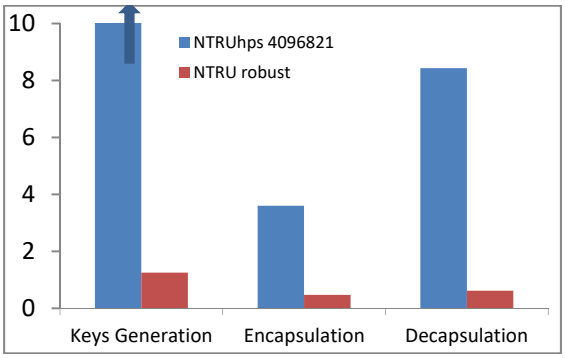

Figure 1: Performance benchmarking between NTRUrobust and NTRUhps4096821. The result values are given in milliseconds (ms).

In this result, we remark that we did better. The speed performance of our *NTRUrobust* compared to the speed performance of *NTRUhps4096821*, is greater by a factor up-to 165 times for key generation function; greater by a factor up-to 8 times for encapsulation function; and greater by a factor up-to 11 time for decapsulation function.

That means that the performance of our release outperforms the *NTRUhps4096821* version for the total cryptographic process (keysGeneration+ Encapsulation+Decapsulation) by a factor up to 93 times faster. This gap performance is essentially due to the performance of our modular multiplication algorithm *FMMA* combined to NTT algorithm.

6.2 Performance benchmarking of *NTRUrobust, Saber, and Kyber*

In this subsection, we present the performance results of our NTRUrobust release compared to the FairSaber and Kyber1024 releases of SABER and KYBER post-quantum KEM schemes , which their parameter sets meets the category 5 security Levels.

Table 2: Performance benchmarking between NTRUrobust, FireSaber, and Kyber1024 releases. The result values are given in milliseconds (ms):

	Keys Gen (ms)	Encap (ms)	Decap (ms)
Kyber1024	0.46	0.63	0.63
FireSaber	2,51	3.12	3.43
NTRUrobust	1,25	0.47	0,62

As illustrates in Table 2 and Figure 3; the NTRUrobust compared to Kyber1024, its performance is slower by factor up to 2.6 times for

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

keys generation, it is faster by factor up to 1.5 for encapsulation, and it has the same performance for the decapsulation. And compared to FireSaber, the performance of NTRUrobust is freater by a factor up 2 times for key generation, more than 6.5 times for encapsulation, and about 5.5 times for decapsulation.

In general cases the keys generation algorithm is not performed for each message exchanged, so it has not the same importance as encapsulation and decapsulation, which are repeated many times for each message exchanged.

That means, in term of performance our NTRUrobust is better than all NTRUhps, KEYBER, and SABER releases.

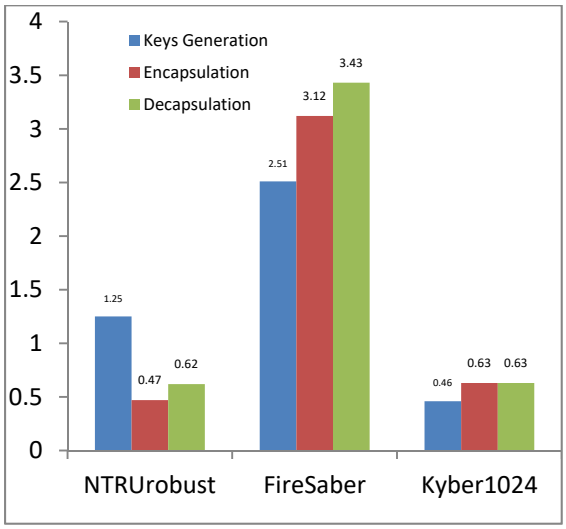

Figure 2: Speed performance benchmarking between NTRUrobust, FireSaber, and Kyber1024. We reportedthe results values to approximate Milliseconds.

7 NTRUrobust Security

Many cryptanalysis works are performed; their principal goal was to check the robustness of the Lattices-Based Cryptography by posing the hardest problems on point lattice in $\mathbb{Z}^n$. The best tools used to prove the security is Lattice reduction by the algorithms (Gram-Schmidt, LLL, BKZ algorithms) and Meet-in-The-Middle attack (MIM)[3] [10][11]. In fact, the mathematicians often estimate the projected security of cryptographic systems by plotting the evolution in "running time" and "space requirements" of the best-known attacks according to level security needed.

For measuring the security level of Lattice-Based Cryptosystems, Martin R. Albrecht et al. developed an estimator tool, which is described in paper under the title "Estimate all the LWE, NTRU schemes"[25]. In this work, we use this tool to estimate the heuristic complexity of BKZ cost model to solve the uSVP (primal attack).

7.1 Security benchmarking

In table 3 below, we use this tool to estimate the security level of NTRUrobust and NTRUhps4096821, Kyber1024, and FireSaber by solving the uSVP (primal attack) with BKZ cost model of size $c=2^{0,292b}$ for the classical security and decreasing this size to $c=2^{0,265b}$ by using the quantum sieving algorithm to consider potential Grover speed-ups[5].

7.1 Security Benchmarking between NTRUrobust, NTRUhps4096821.

In table 3, we present the security benchmarking result between our release NTRUrobust and NTRUhps4096821 release. Our release achieves 2^{230} for classical security level and 2^{208} for quantum security level, whereas the security of NTRU4096821 achieves 2^{195} for classical security level and 2^{175} for quantum security level.

Table 3: Security benchmarking between NTRUrobust, NTRUhps4096821, Kyber1024, and FireSaber .

Scheme\Security	Classical $C=2^{0.292b}$	Quantum $C=2^{0.265b}$
NTRUhps4096821	2^{195}	2^{175}
NTRUrobust	2^{230}	2^{208}
Kyber1024	2^{243}	2^{221}
FireSaber	2^{283}	2^{256}

We remark in this table that the security performance of our release NTRUrobust is more efficient than NTRUhps4096821 by a gain of 35 bits for classical security and 33 bits for quantum security by using respectively the cost models of BKZ $C=2^{0.292b}$, and $C=2^{0.265b}$, when the ***b*** is the block size used the algorithm BKZ to find the uSVP.

According to the result of Albrech et.al estimator, we show that NTRUrobust is more

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

secure than Kyber1024, but it is less secure than FireSaber.

In its report [8240], NIST states that the security of NTRU is based on stronger assumption than LWE or RLWE schemes also based on Lattices [26].

7.2 Decryption failure

The parameters set of NTRUrobust warrants a perfect correctness of the decryption, and the result is obtained by using the script developed by NTRU team [20]. Even we generated the private key in the form $F=pf+1$ [17], as described above in section 5, the result of decryption failure probability is ZERO as presented in Table 4.

Compared to Kyber1024 and FireSaber our release is more confident against an eventual attack using decryption failure [27].

Table 4: Decryption failure comparison

Scheme	Dec. failure rate
NTRUrobust	$2^{-\infty}$
NTRUhps4096821	$2^{-\infty}$
Kyber1024	2^{-174}
FireSaber	2^{-165}

8 CONCLUSION

In this paper, we presented an improved release of NTRUhps4096821 named NTRUrobust. The proposal is an efficient post-quantum key exchange protocol which ensures many interesting security features such as confidentiality of session keys, perfect forward secrecy and a quantum-resistant variant of current classical schemes based on RSA, DH or ECC. We obtained significant enhancements in performance by using our new Fast Modular Multiplication Algorithm with an improved NTT algorithm and also improve the security level and while preserving security compared to NTRUhps4096821, KEYBER and SABER releases. Indeed, the parameters choice and the protocol construction meet the level 5 of the NIST security requirements.

Finally, building more efficient post-quantum protocols with short keys enables practical implementations in real world applications; these constructions are more suitable for applications in various environments in the present and the future.

Our release implementation allows great flexibility, it is very easy to rise the parameters from dimension ***n=1024*** to ***n=2048*** with the same modulus ***q***, and also it is very easy to increase the size of the shared key from ***256 bits*** to ***512 bits***.

REFRENCES:

[1] LilyChen,StephenJordan, Yi-Kai Liu, Dustin Moody, Rene Peralta, Ray Perlner, Daniel Smith, "NISTIR 8105 -Report on Post-Quantum Cryptography", USA, 2016.

[2] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, John Kelsey, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta, Ray Perlner, Angela Robinson, Daniel Smith-Tone, "NISTIR 8309- Status Report on the Second Round of the NIST Post-Quantum Cryptography Standardization Process", NIST, USA, 2020.

[3] Cong Chen, Oussama Danba, Jeffreyrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, "Algorithm Specifications And Supporting Documentation", Wilmington USA, 2019.

[4] El Hassane Laaji, Abdelmalek Azizi, "New Fast Modular Multiplication Algorithm applied to Ring-LWE scheme", Department of Mathematics, Mohammed First University, Oujda, Morocco, 2020.

[5] Alkim, E. Ducas, Poppelman, T. and Schwabe, P. "Post-quantum key exchange-NewHope", Department of Mathematics, Ege University, USA, 2019.

[6] Longa, P. and Naehrig, M, "Speeding up the Number Theoretic Transform for Faster Ideal Lattice-Based Cryptography", Microsoft Research USA, 2019.

[7] Nayuki Project, "Number-Theoric-Transform (Integer DFT)". Link: https://www.nayuki.io/page/number-theoretic-transform-integer-dft

[8] Peter Montgomery, "Modular Multiplication Without Trial Division". Math comput. USA, 1985.

[9] G.V. Assche, G. Bertoni, J. Daemen, P.Peters, and R.Van. "Keccak Hash algorithm". Radboud University,Nederlands, 2016.

[10] J.Hofstein, J. Pipher, and J. H. Silverman: "Introduction Mathematics and Cryptography NTRU" Wilmington USA, 1998.

[11]. J. Hofstein, Jill Pipher, John M. Schanck, Joseph H. Silverman, William Whyte, and Zhenfei Zhang, "Choosing Parameters for NTRUEncrypt" , Wilmington USA 2016.

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

[12] P. Ribenboim. "The New Book of Prime Number Records", New York, Springer-Verlag, 1996.

[13] Regev, O. (2005), "On lattices, learning with errors, random linear codes, and cryptography", In Proceedings of the 37th Annual ACM Symposium on Theory of Computing, pages 84-93.

[14] Lyubashevsky, V. Peikert, C. and Regev, O. (2010). On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, Advances in Cryptology - EUROCRYPT 2010, volume 6110 of Lecture Notes in Computer Science, pages 1-23. Springer.

[15] Eiichiro Fujisaki and Tatsuaki Okamoto, "Secure integration of asymmetric and symmetric encryption schemes", In Advances in Cryptology - CRYPTO99, pages 537–554, 1999. https://link.springer.com/chapter/10.1007/3-540-48405-1-34. 5, 11, 20

[16] Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrède Lepoint, Vadim Lyubashevsky, John M. Schanck, Peter Schwabe, Gregor Seiler, Damien Stehlé, "CRYSTALS-Kyber Algorithm Specifications And Supporting Documentation (version 2.0)", NIST – Wilmington USA March 30, 2019.

[17] Jan-Pieter D'Anvers, Angshuman Karmakar, Sujoy Sinha Roy, and Frederik Vercauteren, "Saber: Module-LWR Based Key Exchange, CPA-Secure Encryption and CCA-Secure KEM", in AFRICACRYPT 2018, pages 282-305, 2018.

[18] Jan-Pieter D'Anvers, Angshuman Karmakar, Frederik Vercauteren "SABER: Mod-LWR based KEM (Round 2 Submission)", imec-COSIC, KU Leuven, Belgium, 2019.

[19]. Cong Chen, Oussama Danba, Jeffreyrey Hoffstein, Andreas Hulsing, Joost Rijneveld, John M. Schanck, Peter Schwabe, William Whyte, Zhenfei Zhang, "Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU" Wilmington USA 2019.

[20] Mohan Rao Mamdikar, Vinay Kumar and D. Ghosh, " Enhancement of NTRU public key" National Institute of Technology, Durgapur 2013.

[21] John Scham and NTRU team, "Decryption failure script" link: https://github.com/jschanck/ntru-ephem-dfr.

[22] Nick Howgrave-Graham, Phong Q. Nguyen, David Pointcheval, John Proos, Joseph H. Silverman, Ari Singer, and William Whyte, "The Impact of Decryption Failures on the Security of NTRU Encryption", NTRU Cryptosystems Burlington, CNRS France, University of Waterloo, Canada.

[23] El Hassane LAAJI, abdelmalek AZIZI, Link Google drive of our NTRUrobust implementation:
https://drive.google.com/file/d/1Cbe0fTFphfxzEvMCLTQjdIerlaEsc6cm/view?usp=sharing
Mohammed first University Oujda, Morocco, 2020.

[24] NIST link website of NTRU, SABER, and KYBER cryptosystem implementations. https://csrc.nist.gov/projects/post-quantum-cryptography/round-2-submissions

[25] Martin R. Albrecht, Benjamin R. Curtis, Amit Deo, Alex Davidson, Rachel Player, Eamonn W. Postlethwaite, Fernando Virdia, and Thomas Wunderer, "Estimate all the {LWE, NTRU} schemes", In Security and Cryptography for Networks - 11th International Conference, SCN 2018, volume 11035 of Lecture Notes in Computer Science, pages 351-367. Springer, 2018.

[26] Gorjan Alagic, Jacob Alperin-Sheriff, Daniel Apon, David Cooper, Quynh Dang, Yi-Kai Liu, Carl Miller, Dustin Moody, Rene Peralta Ray Perlner, Angela Robinson, Daniel Smith-Tone, "Status Report NISTIR 8240 on the First Round of the NIST Post-Quantum Cryptography Standardization Process", USA 2019.

[27] Daniel J. Bernstein1, "Comparing proofs of security for lattice-based encryption", Department of Computer Science, University of Illinois at Chicago, Chicago, IL 60607 - 7045, USA2, Horst Gortz Institute for IT Security, Ruhr University Bochum, Germany djb.at.cr.yp.to